

---

---

# Introducción a Ruby

Herencia

---

---

- El operador | |
- Métodos de clase
- Variables de clase
- Herencia

# var = var || something

- El operador || **evalúa el lado izquierdo** de la expresión:
  - Si es **true** - lo **retorna**
  - **Sino** - **retorna el lado derecho** de la expresión
- **@x = @x || 5** retornará **5** la primera vez y **@x** la siguiente
- La forma corta:
  - **@x || = 5**

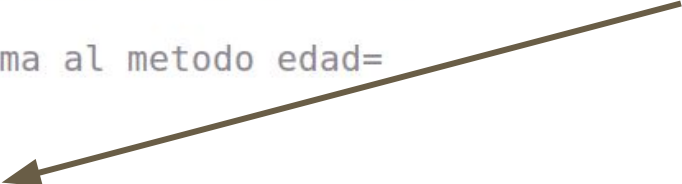
# var = var || something

```
class Persona
  attr_reader :edad
  attr_accessor :nombre

  def initialize (nombre, edad) # CONSTRUCTOR
    @nombre = nombre
    self.edad = edad # llama al metodo edad=
  end
  def edad= (new_edad)
    @edad ||= 5 # default
    @edad = new_edad unless new_edad > 120
  end
end

personal = Persona.new("Karen", 130)
puts personal.edad # => 5 (default)
personal.edad = 10 # Cambiar a 10
puts personal.edad # => 10
personal.edad = 200 # Tratar de cambiar a 200
puts personal.edad # => 10 (Se mantiene el valor)
```

Solamente  
setea a 5 la  
primera vez!



# Métodos y Variables de Clase

- Se **invoca** sobre la **clase** (en oposición a las instancias)
  - Similar a los métodos **static** de Java.
- Se utiliza **self** para definir que un método es de clase y no de instancia.
  - **self** fuera de un método se refiere a un objeto de clase

# Métodos y Variables de Clase

- Existen tres formas de definir métodos de clase en Ruby:
  - **self.nombre\_metodo**
  - `def nombre_metodo << self`
  - **Clase.nombre\_metodo**
- Las variables de clase empiezan con @@

# Creación de métodos de clase

```
class Matematica
  def self.doble(var) # 1. Utilizando self.
    cnt_llamadas; var * 2;
  end
  class << self # 2. Utilizando << self
    def cnt_llamadas
      @@cnt_llamadas ||= 0; @@cnt_llamadas += 1
    end
  end
end

def Matematica.triple(var) # 3. Fuera de la clase
  cnt_llamadas; var * 3
end
```

# No existen instancias creadas!

```
puts Matematica.doble 5 # => 10
puts Matematica.triple(3) # => 9
puts Matematica.cnt_llamadas # => 3
```

# Herencia de Clases

- Toda clase hereda **implícitamente** de **Object**
  - **Object** mismo hereda de **BasicObject**
- **No** existe la herencia múltiple
  - Se utilizan **mixins** para simular herencia múltiple



# Herencia

```
class Perro
  def to_s
    "Perro"
  end
  def bark
    "Ladra fuerte"
  end
end
```



```
class PerroChico < Perro
  def bark # Override
    "Ladra despacio"
  end
end
```

```
perro = Perro.new
perro_chico = PerroChico.new
puts "#{perro}1 #{perro.bark}" # => Perro1 Ladra fuerte
puts "#{perro_chico}2 #{perro_chico.bark}" # => Perro2 Ladra despacio
```

## Entonces..

- La herencia permite **sobreescribir** el comportamiento de la clase padre.
- Los métodos de clase **no necesitan** una instancia de objeto para ser **invocados**.
- Las variables de clase inician con @@